Learning to Form Coalitions in Heterogeneous Teams from Suboptimal Demonstrations

Rachit Bhargava

Georgia Institute of Technology

Undergraduate Research Thesis

May 7, 2021

Dr. Harish Ravichandar - Mentor	Dr. Sonia Chernova - Second Reader
DocuSigned by: Harish Ravichandar 8FB0C7042D8C4EC...	DocuSigned by: Sonia Chernova A408D608A45E4A8...
Signature	*Signature*
5/7/2021 | 9:47 AM EDT	5/7/2021 | 10:21 AM EDT
Date	*Date*

Abstract

Multi-agent systems (MAS) have proven to be effective in a wide range of domains including warehouse automation, defense, agriculture, and environmental modeling. Heterogeneous MAS, often made up of a different types of agents with complementary capabilities, can particularly be effective in handling complex scenarios that require a variety of skills. However, coordinating such teams presents significant challenges that require either experts with near-perfect domain knowledge or learning approaches that require vast amounts of computation resources. This thesis explores the possibility of learning to coordinate heterogeneous MAS from humans who might not be experts and will not act optimally as a result. Specifically, we develop a framework that can learn to form effective coalitions (an instance of the task allocation problem) that can solve multiple concurrent tasks from suboptimal demonstrations. To this end, we first learn to predict the reward associated with a given allocation using supervised learning, and subsequently optimize over the space of allocations to identify one that will maximize the predicted reward. As such, we effectively utilize non-expert data to bootstrap learning, instead of attempting to learn from scratch. Consequently, our framework neither requires considerable domain knowledge nor incurs an unsustainable amount of computational burden. In order to implement and evaluate our framework, we also contribute a user study interface capable of collecting demonstrations from remote users as they play a virtual multi-agent game designed using the StarCraft II simulator. Our experimental results demonstrate that we can learn the reward functions associated with the tasks with an accuracy of over 70% while having access to just the suboptimal demonstrations. Reward functions learned from such data can then be used to predict an ideal assignment to get better performance than what has been seen in the training data.

1 Introduction

The problem of coordinating a set of robots (or agents) in order to perform a set of tasks naturally appears in a variety of applications, such as agriculture (Kim & Son, 2020), warehouse automation (Claes, Oliehoek, Baier, Tuyls, et al., 2017), defense (TREATY, 2008), and environmental monitoring (Ma, Ma, Liu, & Sukhatme, 2018). The problem of appropriately assigning these agents to the tasks to maximize the agent productivity and achieve successful completion of tasks is known as the multi-robot task allocation (MRTA) problem (Gerkey & Matarić, 2004). MRTA is crucial for multi-agent collaboration due to the impracticality to complete the tasks with inappropriate agent assignments (Korsah, Stentz, & Dias, 2013). There can be cases when there are some operations that need to be carried out to complete a mission with finite number of agents (for example, deploying robots in warehouses) or when tasks have certain restrictions that need to be met (for example, a task to carry 200 lb logs of wood).

Past studies suggest a range of ways to deal with MRTA problem based on the amount of information available. If we consider the case of having sufficient agents to complete the task, an approach by Lau and Zhang (Lau & Zhang, 2003) casts this problem as an optimization problem requiring agents to be matched to the tasks. Other studies, along the similar lines, introduce approaches such as evolutionary algorithms (Vidal, Crainic, Gendreau, Lahrichi, & Rei, 2012), constraint-programming-based algorithms (Shaw, 1998), auction-based algorithms (Gerkey & Mataric, 2002; Wicke, Freelan, & Luke, 2015) and multiarmed bandit algorithms (Hassan & Curry, 2014) to solve it. While these approaches may be useful for cases when any information other than the available agents is not available, they do not use the extra information even when it is available. This extra information can be in the form of having access to agent ability charts (for example, amount of weight they can carry, and how fast can they move) or even the task requirements (for example, how heavy is the log that needs to be carried, and how fast should a patient be delivered to the hospital).

For this study, we consider agent-wise ability information being available along with the data about how well the agents performed with certain assignments for a range of tasks in the past (referred to as demonstrations later in the paper). Demonstrations, in our case, are the records of agent assignment information to the available tasks. Most of the existing approaches to task allocation are, however, not designed to consider the availability of demonstrations.

Learning how to do multi-agent task allocation with the provided demonstration information instead of solving the problem by searching for a solution with no additional information from human experts is usually more efficient especially because the search space tends to increase with more number of agents available and number of tasks to complete due to increasing number of possible assignments that can be done. Given the type of information available, behavior cloning and inverse reinforcement learning (Arora & Doshi, 2021) are two of the most popular approaches that can be used to solve the MRTA problem with some information because of their ability to either mimic the behavior that led to a success (as seen in behavior cloning) or their ability to extrapolate enough information that normal reinforcement learning techniques can solve the problem (Chen, Paleja, & Gombolay, 2020).

Behavior cloning (Hwang, Chen, & Wu, 2012), however, has a few limitations. First, it is likely to mimic the assignments done to get another assignment for some new problem. While this might be the behavior needed if the available demonstrations are all optimal, it fails to provide good results if the available demonstrations are not optimal (Brown, Goo, Nagarajan, & Niekum, 2019). Also, the algorithm tends to be unreliable when the order of demonstrations is not fixed (Codevilla, Santana, López, & Gaidon, 2019).

Inverse reinforcement learning (Abbeel & Ng, 2004) attempts to infer a reward function, for the normal reinforcement learning algorithm to use, from the available information about an agent's performance from previously recorded demonstrations. This method also shares an issue with behavior cloning; it can provide results that are just as good as

the best demonstration available. Unfortunately, high-quality demonstrations are hard to obtain. Sometimes, demonstrations from only the non-experts, or people who do not know how to do the tasks optimally, might be available. In some scenarios, even the experts might struggle to perform tasks optimally. This can often result in the agent receiving a series of suboptimal or a mixture of suboptimal and optimal demonstrations to learn from, which in turn causes the inferred reward function to be inaccurate.

Human-Agent Transfer (Taylor, Suay, & Chernova, 2011) is another approach that suggests using reinforcement learning to learn from demonstrations by combining it with transfer learning and known learning from demonstrations techniques to train an agent by starting from the set of available demonstrations, but it mainly serves to speed up the learning phase of reinforcement learning. The technique requires a reward function, however, which is again not available to us.

To eliminate the need for having a known reward function, a recently introduced inverse reinforcement learning algorithm, T-REX, extrapolates the user's intent from past demonstrations to then infer the reward function that agrees with the extrapolated demonstrator's interests (Brown et al., 2019). This inferred reward function can then be used with reinforcement learning techniques to get to a better assignment than what has already been seen.

In this work, we develop an algorithm to learn multi-agent task allocation from suboptimal demonstrations and reinforcement. Specifically, much like T-REX, we first learn the task requirements by inferring the reward or score functions and then optimize over the assignments space to fulfill those requirements. This work can be used in solutions to ST-MR-IA (Korsah et al., 2013) problems (or in other words, problems involving multiple robots that are capable of handling one task at a time where we need to assign them to the tasks in the beginning) to learn from suboptimal demonstrations and then optimize the agent assignments to get a better performance than what is observed in the available demonstrations.

2 Literature Review

This proposal shall divide the literature review in two parts, first for multi-agent task allocation, and second for learning algorithms.

2.1 Multi-Agent Task Allocation

This part of the literature review discusses some of the progress made to solve the multi-agent task allocation problem related to our work. Before we start, however, it is important to define our problem space. A survey from Gerkey (Gerkey & Matarić, 2004) suggests to divide the multi-agent task allocation problems depending on the kind of tasks, agents available and the assignment required. On the basis of this theory, our work shall focus on the ST-MR-IA problems. This can also be seen as focusing on MT-SR-IA problems because of ST-MR-IA problems being mathematically equivalent to it with the roles of tasks and robots reversed (Korsah et al., 2013).

A study (Lau & Zhang, 2003) suggests to solve all the ST-MR-IA problems, provided that we have sufficient agents to complete the tasks, by considering them as optimization problems and using the greedy algorithm approach to solve the problem. Some other methods like evolutionary algorithms (Vidal et al., 2012) and constraint-programming-based algorithms (Shaw, 1998) also agree with the first claim of Lau's survey, but all these approaches fail to suggest a way to get information from the past demonstrations, when such data is available. In other words, these approaches suggest to explore the entire list (or space) of possible allocations and report the allocation that give us the best results. This is often not possible because of the unavailability of a reliable simulator and a known reward function. Even if they are available, these approaches would be impractical because of the combinatorially increasing number of allocation possibilities with increasing number of robots available and tasks to complete.

Another approach is the auction-based algorithm (Gerkey & Mataric, 2002; Jones

et al., 2006; Shiroma & Campos, 2009) where the individual agents auction their feasibility to work on a task and the best bidder is chosen after every time interval to determine the robot that gets to work on the task. This approach, while it allows agents to learn from past information, does not leave any room for errors. The algorithm may not work as intended if there is an error in the bids placed. This can happen when the agent bidding does not know the feasibility to work on a task. This approach also does not leave much room for collaboration because the best bidder gets to work on the task by himself. This would make this approach impractical in cases when multiple agents are needed to complete some given task.

A study by Wicke (Wicke et al., 2015) suggests using bounty-hunter algorithm to solve the multi-agent task allocation problem. The study suggests to have a bondsman agent that keeps track of all the tasks. The worker agents can express their availability to the bondsman agent when tasks are available. While this sounds like a promising technique, it does not allow collaboration on any given task (meaning that only one agent can do a task at any given moment). Because of this limitation of Wicke's algorithm's, it might often be impossible to complete tasks that needs more than one agent working on them at the same time.

Another approach (Langford & Zhang, 2007) introduces a variation of the multiarmed bandit problem to handle the assignment of agents to tasks. The work adds the requirement of having some contextual information about the agents (which can be the state of the agents or information about all the agents in problem) and uses this information to make a better assignment. However, it does not suggest how we can use the ranked demonstrations or assignments information to make a better assignment decision. Instead, it simply ignores this extra information and makes a decision based solely on the available information about the agents. In other words, this approach does not learn anything from the other demonstrations.

In this study, we aim to use the information from suboptimal demonstrations to

get to better-than-demonstration assignment without having any information of the reward function. We can do this by learning a way to do better than other available demonstrations. We can then use the learned model to then get the optimal assignments. The next section will talk about the learning algorithms that can potentially be used.

2.2 Learning Algorithms

This part of the literature review discusses the research done in the field of inverse reinforcement learning to get better-than-demonstrator performance from the provided demonstrations. Most of the past work (Abbeel & Ng, 2004; Ziebart, Maas, Bagnell, & Dey, 2008) assumes that the provided demonstrations would be optimal, thus suggesting inverse reinforcement or imitation learning techniques to solve the problem. Others (Argall, Chernova, Veloso, & Browning, 2009; Gao et al., 2018) have some tight constraints like having access to information about permissible actions and their inferences on partial reward function.

Early research in this field (Pomerleau, 1991) aimed at mimicking the behavior seen in the provided demonstrations (imitation learning). While this might have kickstarted the research in the field of learning from demonstrations, it was later proven that the approach had the constraint of continuously getting feedback from a human and applying corrections progressively, without which there might be large generalization error (Ross, Gordon, & Bagnell, 2011). Imitation learning has also teamed up with deep learning to solve the problem of learning from demonstrations in a recent study (Ho & Ermon, 2016), but these approaches often tend to be computationally expensive and not practical for usage in real-world problems. Another limitation is that they tend to mimic what they observed in the demonstrations, which implies that suboptimal demonstrations would result in suboptimal results.

To circumnavigate the issue of having to deal with the generalization error, some

studies (Finn, Levine, & Abbeel, 2016; Ziebart et al., 2008) have proposed to use inverse reinforcement learning to infer a reward function from the provided demonstrations so that states that have not yet been visited can be tested. This approach, however, assumes that all the provided demonstrations are optimal. Because of this, these techniques cannot provide performance better than the demonstrations themselves. Another study (Ziebart et al., 2008) also suggests to maximize the entropy of the final policy at the main reinforcement learning stage after the reward inference to minimize impact by some suboptimal demonstrations, but the technique becomes unusable when dealing with episodic memory of a majority of suboptimal demonstrations.

When it comes to learning from suboptimal demonstrations, the field saw major progress, starting 2011. The first major work in this field (Grollman & Billard, 2011) suggested a way to learn from failed demonstrations. However, this approach needs manual sorting of demonstrations into failed, over-shot and under-shot demonstrations, each of which require proper knowledge of ground truth goal or reward function. These functions might not be feasibly available in real-world scenarios. Later studies (Choi, Lee, & Oh, 2019; Zheng, Liu, & Ni, 2014) focused on increasing tolerance of models to suboptimal demonstrations, but none of them addressed cases with more than 50% tolerance, meaning that they need at least half of the provided demonstrations to be optimal.

T-REX (Brown et al., 2019) uses ranked demonstrations as a way to extrapolate a reward function that fits the rankings. While this approach does infer a reward function that can be used for better performance, the results are unreliable, meaning that they are not guaranteed to be better than the provided suboptimal demonstrations. Also, the errors are susceptible to become magnified if the demonstrations have some changes that has little to no impact on the output. D-REX (Brown, Goo, & Niekum, 2020), on the other hand, proposes introduction of errors in the provided demonstrations and assumes that introducing error by

result in worse demonstrations and then uses T-REX to extrapolate a reward function. This approach also indirectly assumes that you have an inexpensive simulator available for the task at hand, which is often not true. In real-world scenarios, using D-REX would also not guarantee results since introducing errors may sometimes result in better demonstrations, which may cause the trained model to be worse than the given demonstrations.

This work aims at exploring the field of learning from suboptimal demonstrations and checking its feasibility for learning task-agent assignment as one of its applications. This will allow us to propose a solution to the assignment problem just from some prior knowledge of the performance of agents involved. Inspired by a recent work (Chen et al., 2020) that tends to enhance some existing methods like T-REX, we introduce a score metric to properly organize the demonstrations instead of labeling them with just a rank.

3 Problem Formulation

Given N demonstrations for assignments for M tasks along with distribution of P traits (or a measure of capabilities) for the U different types of agents in the form of $\mathcal{D} = \{X_i,\ Q_i\}_{i=1}^{N}$ (where X_i is the agent-task distribution matrix (assignments) of the shape M tasks $\times U$ types of agents for the i-th demonstration and Q_i is agent-trait distribution matrix (measure of capabilities of each agent) of the shape U agents $\times P$ traits for the i-th demonstration) and task-wise performance scores of the teams observed in the demonstrations $\mathcal{S} = \{s_i^T\}_{i=1}^{N}$ (where s_i is the task-wise score vector for the i-th demonstration), we are interested in deciphering a reward function for each task $\hat{\mathcal{R}} = \{\hat{r}_m\}_{m=1}^{M}$ that determines any team's performance for that task. The objective is to learn a mapping from the trait requirement matrices ($Y = XQ$) to the scores. This will allow us to reduce our problem to an optimization problem which we can solve to get the ideal agent distribution among the tasks that need to be tackled (let us call this ideal agent distribution matrix, $\mathbb{X}^*$) from available agents N_S

(N_S being a 1-dimensional vector of size U) and the new agent-trait distribution matrix Q'. We can then work in the agent space to get the distribution that satisfies the ideal trait distribution. This will enable us to provide a mechanism to assign agents in a way to achieve performance better than any of the demonstrators. While we could convert the agents assignment space (number of agents assigned to each task) to an action space that can be used by traditional inverse reinforcement learning algorithms, the results would be valid just for the observed tasks and would not be easily generalizable to the other tasks.

4 Approach

We approach this problem by treating it as two sub-problems: i) to learn to predict the reward function (or in other words, the score function) associated with the assignments, task-wise ($\hat{\mathcal{R}} = \{\hat{r}_m\}_{m=1}^{M}$), ii) and to optimize over allocations such that the predicted reward is maximized. The reward function for each task is given by $r_m : \mathcal{Y} \rightarrow \mathbb{R}^+$, where $\mathcal{Y}$ represents the space of all possible trait distribution matrices. We estimate $\mathbb{X}^*$ by maximizing the reward. To this end, we solve a constrained maximization problem as follows, subject to the constraints of not allocating more than the available agents in our team.

As mentioned earlier, our suggested solution to the given problem would involve dividing the process in two steps.

4.1 Step 1: Infer the reward functions

Given a set of demonstrations $\mathcal{D} = \{X_i, Q_i\}_{i=1}^{N}$, we can calculate the set of trait-distribution matrices (Y) such that, $\mathbf{Y} = \{Y_i | Y_i = X_i Q_i, 1 \leq i \leq N\}$. It must be noted that each Y_i matrix contains the distribution of traits for each task. As a result, we use the trait distribution matrices to train neural networks for each task to predict the score with the trait distribution. Each neural network is a single hidden layer network consisting of five neurons with sigmoid (Narayan, 1997) as the activation function. The output of a task's

neural network, with the hidden layer's neuron h having the weight w_h and bias b_h, the final neuron having the weight w_0 and bias b_0 and activation function ϕ, would then be the predicted score $\hat{s}_m$ calculated as follows.

$$\hat{s}_m = w_0\phi(w_h Y_i + b_h) + b_0 \tag{1}$$

This allows us to find a mapping, or in other words, infer the reward functions, from the $\mathcal{Y}$ space to a single reward value in $\mathbb{R}^+$ for each task.

Figure 1: Accuracy Distribution

4.2 Step 2: Optimize the inferred reward functions

Once we have the inferred reward function, we calculate the optimized task-trait distribution matrix by solving a constrained optimization problem so that we maximize the sum of the

reward functions with the available agents. We can use the total number of agents of each species available, N_S, as a constraint beyond which we do not allow more agents from any given type to be assigned. We solve the constrained-optimization problem using the Sequential Least Squares Programming algorithm (Kraft et al., 1988). This can be written as follows.

$$X^* = \arg \max_{X \in Z^+_{M \times U}} \sum_{m=1}^{M} \hat{r}_m(X_m Q') \tag{2}$$

$$s.t., \; X^{*T} \cdot \mathbb{1} \preceq N_S \tag{3}$$

5 Experimental Validation: StarCraft II

In our game-world experiment, we designed a StarCraft II Map that involved battling enemy units by splitting the available agents in several teams. The objective is to collect demonstrations from the human users about how they would do assignments with the information about the tasks and agents.

5.1 Study Design

The simulator is developed in Python 3.8.0 and uses Flask web framework to provide an interface that can be used to interface with the game and aid with data collection. The data is collected in CSV sheets which store both the user tokens and their assignment information. The application uses BurnySC2 StarCraft II interface library instead of the general PySC2 interface library because of the abstracted functions available that takes away the need to manually handle the low-level controls of the agents.

5.1.1 Game Map

We use a battle scenario game map for experimental validation of our approach. The user has the job of dividing a cluster of 16 Zealots and 4 Stalkers into two teams. The first team

would fight a battalion of 24 Zerglings and the other team would fight a battalion of 11 Zerglings and 4 Roaches. The objective is to win against both the enemy battalions (or in other words, kill all the enemy units) as quickly as possible.

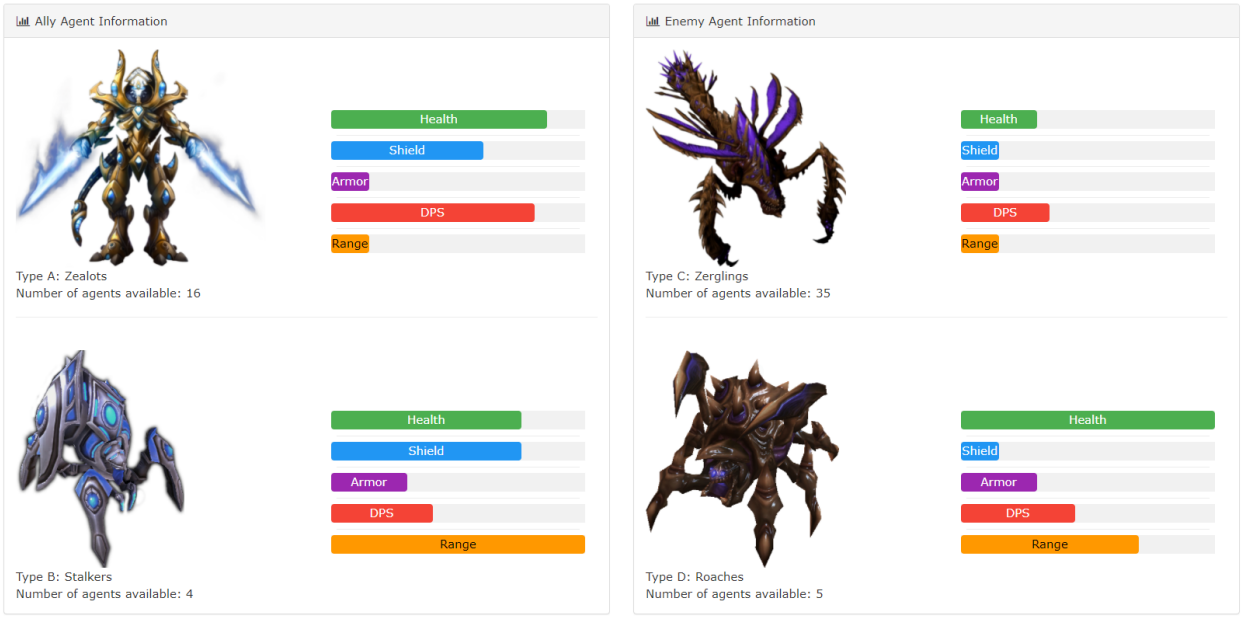


Figure 2: Agent-Trait Distribution Chart

Here, zealots, stalkers, zerglings and roaches are names of units in StarCraft II. The player could also optionally choose to not use all the units by leaving some of the units to stand ground in case some enemy unit attempts to cross the bridge.

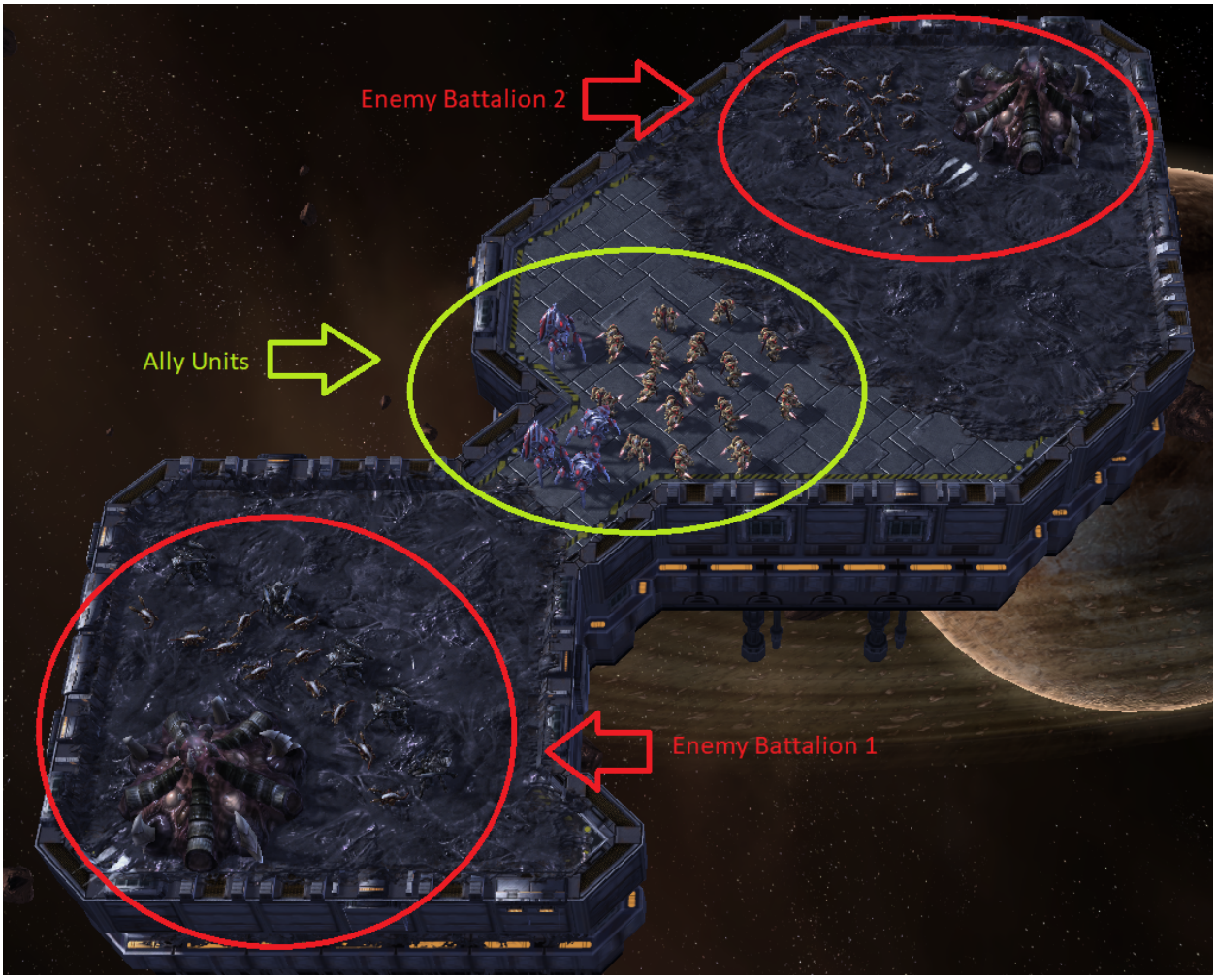


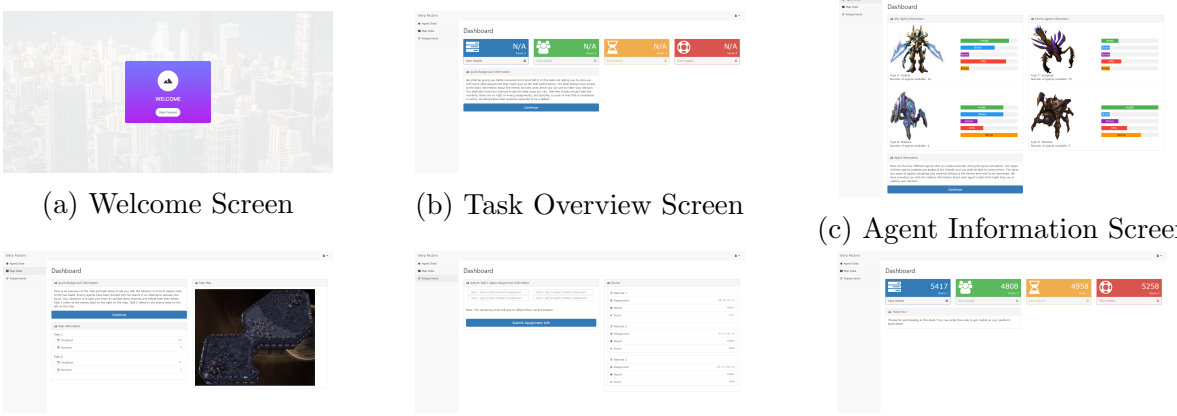
Figure 3: Labeled Map 1 Screenshot from StarCraft II Map Editor

5.1.2 User Flow

The users are given relative information about the ally (the agents) and enemy units' health, shield, armor, damage per second capability and attack range (the traits). They are also provided with the task information in the form of division of enemy units (task requirements). They are then asked to form teams (or provide allocations) to fight the enemy units without leaving any units at base. Once done, they are provided access to the battle simulation and the final scores they received for the combination of tasks (overall) and the individual tasks. This allows us to simulate receiving suboptimal demonstration for every instance where the user's assignment resulted in a victory but not the highest possible score.

Our simulator allows users to provide assignment information for the given experiments without having to install StarCraft II on their devices. This also allows us to run the games in the same environment every time.

To summarize, the overall task for the users involves, getting redirected to our StarCraft II simulator, getting acquainted with the game, the tasks and the experiment itself, and attempting to win the game with the assignments that they think would be ideal. The users are given three additional attempts to provide new assignments based on the past scores they received.



(a) Welcome Screen (b) Task Overview Screen (c) Agent Information Screen

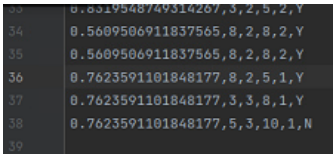
(d) Task Layout Screen (e) Score Display Screen (f) Thank You Screen

Figure 4: StarCraft II Simulator User Interface

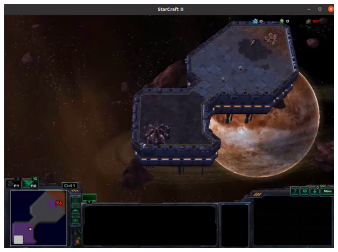
5.1.3 Information Flow

Once a user enters the assignment information, it gets stored in a CSV file with a marker which says that the assignment has not been simulated yet. On a separate thread, a script checks this file for any unsimulated assignments every five seconds. As soon as it notices this new assignment, it picks it up, simulates the assignment, first overall, then separately for each task, and marks the assignment as simulated. After each simulation, the script also stores the results in a separate CSV file and creates a replay file that is later used by the simulator to show to the user what might have happened in the battle. The CSV file can be used by the simulator to display the results to the user and also by us to get the required

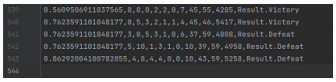
information about the demonstrations.



(a) Data from User Input



(b) StarCraft II Simulator



(c) Data from StarCraft II Simulation

Figure 5: User assignment simulation on server

5.1.4 Reward Design

The reward function used for in our experiments is as follows. It must be noted that the game times out after 60 seconds of gameplay, meaning that the user's assignment must achieve victory within 60 seconds. We decided on this time limit because the units always stop interacting with their surrounding after about 50 seconds of gameplay for our specific setup. It must also be noted that the number of enemies killed in the following equation refers to the number in task m.

$$r_m = \begin{cases} 120 + 5 \cdot enemies\_killed_m - 2 \cdot time\_taken + 25, & \text{if all enemies are defeated} \\ 120 + 5 \cdot enemies\_killed_m - 2 \cdot time\_taken, & \text{if all enemies are not defeated} \end{cases}$$

Simulating all possibilities (see the graph below) for this experiment shows that there is a 31% chance of victory with a random assignment and that the final score for victory can vary from 328 to 361 which shows that there is a possibility to learn from suboptimal demonstrations. Task-wise score distribution can be seen in Figure 6.

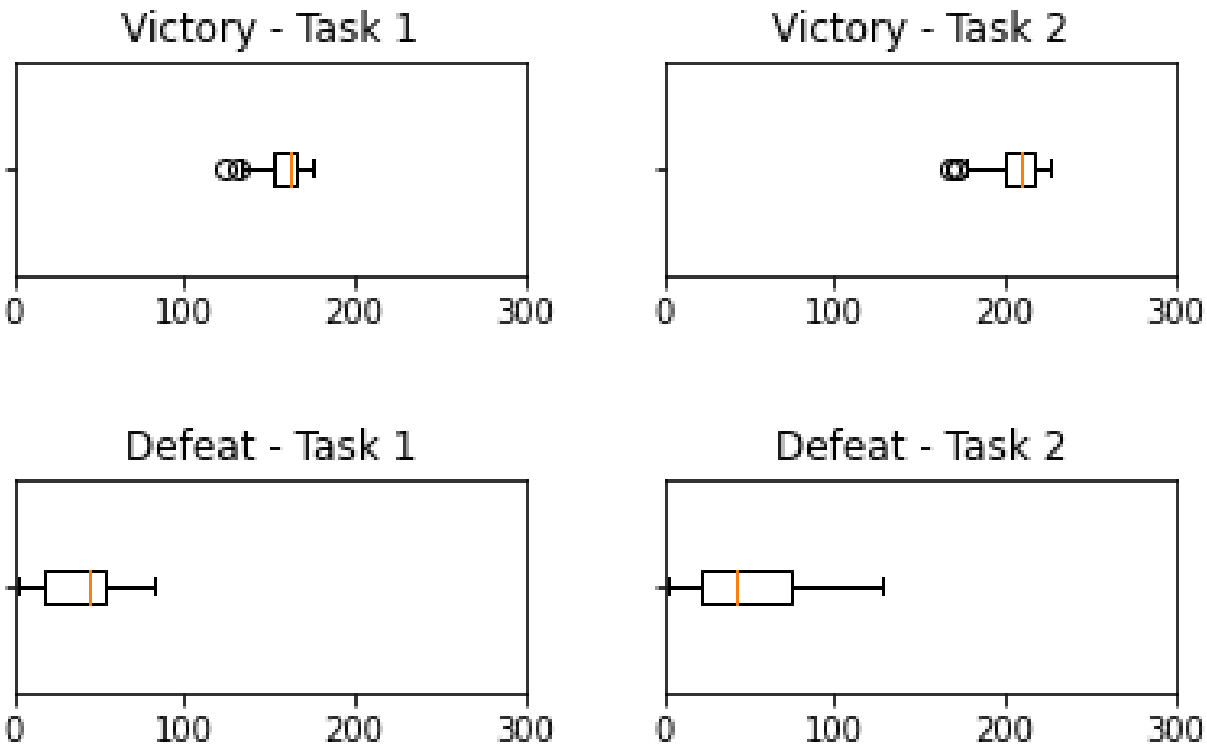


Figure 6: Game Map Score Distribution from Random Assignment

5.2 Evaluation

In this section, we provide the details associated with our experimental evaluation based in the experimental platform detailed above.

5.2.1 Data

For the purpose of experimental evaluation, we gathered performance data after simulating complete assignments (no agent was left unassigned to the tasks available). The data disclosed included the exact assignment of the agents to each task (X matrix), the capabilities of each agent (Q matrix) and task-wise score for each demonstration ($\mathcal{S}$). To simulate the condition of having suboptimal demonstrations, we sorted the demonstrations in descending order of the sum of scores for all the tasks. Finally, we used the worst 80% demonstrations of the dataset to train the neural networks so that they may infer the reward function for

each task and held the remaining 20% as the testing dataset.

5.2.2 Reward Inference

We first verify the hyperparameters of the neural network using k-fold cross validation with $k = 10$. Doing so on a dataset of 85 data points results in an average accuracy of 68% for Task 1 and 66% for Task 2 for the training dataset and 80% for Task 1 and 93% for Task 2 for the testing dataset for Task 2. The error distribution can be seen in Figure 1.

Running Step 1 of the algorithm with the suggested network setup results in the reward function having an accuracy of about 95% for Task 1 and 91% for Task 2 for the training dataset and 73% for Task 1 and 90% for Task 2 for the testing dataset. Further analysis of the inferred score function shows us that the network consistently underestimates the score because of the fact that it is trained completely on the suboptimal demonstrations which included the demonstrations that did not result in a victory. Due to the design of the current reward function, the high values of the score can be mostly found in the near-optimal victory demonstrations that are located in the testing dataset.

5.2.3 Optimizing Allocation

Figure 7 compares the true and predicted scores for each assignment in the training and testing dataset. Step 2 of the algorithm tends to find the allocation with the maximum value on the y-axis (predicted score) while the actual best allocation is the one with the maximum value on the x-axis (true score).

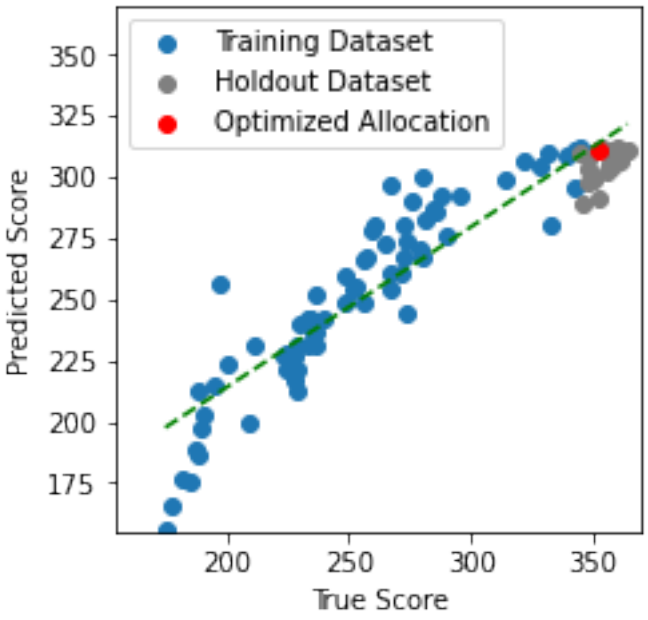


Figure 7: Score Distribution - True vs Predicted

Running Step 2 of the algorithm using the inferred reward functions from the last step and the initial state of no assignment of agents to tasks results in the assignment of $[[8, 2], [8, 2]]$ (assignment 8 zealots and 2 stalkers to Task 1 and 8 zealots and 2 stalkers to Task 2) which also happens to lie in the top 10% scores using the ground truth reward functions (see Figure 7). The suggested assignment is also close to $[[8, 1], [8, 3]]$ which is the assignment with the highest score using the ground truth reward functions.

6 Conclusion

This thesis investigated the possibility of learning to form coalitions within heterogeneous MAS from suboptimal demonstrations. Our investigation revealed the promise in learning to predict the quality of different allocations. Specifically, a simple supervised learning approach was capable of predicting the reward function associated with a given allocation. Further, once trained, this predictive network can be used to optimize over all possible allocations. We found that such optimization can indeed result in an allocation that results in a higher

reward than allocations found in the demonstration set. A key limitation of this work is that the approximation errors made by our predictive model are ignored and propagated downstream. Our future work will address this concern by viewing the predictive model as a prior belief over rewards and which can be used to bootstrap bandit-based approaches that can rely on environmental interactions to "fine tune" our understanding of what coalitions will yield the highest rewards.

References

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on machine learning* (p. 1).

Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, *57*(5), 469–483.

Arora, S., & Doshi, P. (2021). A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 103500.

Brown, D. S., Goo, W., Nagarajan, P., & Niekum, S. (2019). Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *Proceedings of thirty-sixth international conference on machine learning* (pp. 783–792).

Brown, D. S., Goo, W., & Niekum, S. (2020). Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Proceedings of 3rd conference on robot learning* (pp. 330–359).

Chen, L., Paleja, R., & Gombolay, M. (2020). Learning from suboptimal demonstration via self-supervised reward regression. In *Conference on robot learning (corl '20).*

Choi, S., Lee, K., & Oh, S. (2019). Robust learning from demonstrations with mixed qualities using leveraged gaussian processes. *IEEE Transactions on Robotics*, *35*(3), 564–576.

Claes, D., Oliehoek, F., Baier, H., Tuyls, K., et al. (2017). Decentralised online planning for multi-robot warehouse commissioning. In *Aamas'17: Proceedings of the 16th interna-*

tional conference on autonomous agents and multiagent systems (pp. 492–500).

Codevilla, F., Santana, E., López, A. M., & Gaidon, A. (2019). Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the ieee international conference on computer vision* (pp. 9329–9338).

Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning* (pp. 49–58).

Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S., & Darrell, T. (2018). Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*.

Gerkey, B. P., & Mataric, M. J. (2002). Sold!: Auction methods for multirobot coordination. *IEEE transactions on robotics and automation*, *18*(5), 758–768.

Gerkey, B. P., & Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research*, *23*(9), 939–954.

Grollman, D. H., & Billard, A. (2011). Donut as i do: Learning from failed demonstrations. In *2011 ieee international conference on robotics and automation* (pp. 3804–3809).

Hassan, U. U., & Curry, E. (2014). A multi-armed bandit approach to online spatial task assignment. In *2014 ieee 11th intl conf on ubiquitous intelligence and computing and 2014 ieee 11th intl conf on autonomic and trusted computing and 2014 ieee 14th intl conf on scalable computing and communications and its associated workshops* (pp. 212–219).

Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in neural information processing systems* (pp. 4565–4573).

Hwang, K.-S., Chen, Y.-J., & Wu, C.-J. (2012). Fusion of multiple behaviors using layered reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *42*(4), 999–1004.

Jones, E. G., Browning, B., Dias, M. B., Argall, B., Veloso, M., & Stentz, A. (2006). Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks.

In *Proceedings 2006 ieee international conference on robotics and automation, 2006. icra 2006.* (pp. 570–575).

Kim, J., & Son, H. I. (2020). A voronoi diagram-based workspace partition for weak cooperation of multi-robot system in orchard. *IEEE Access*, *8*, 20676–20686.

Korsah, G. A., Stentz, A., & Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, *32*(12), 1495–1512.

Kraft, D., et al. (1988). A software package for sequential quadratic programming.

Langford, J., & Zhang, T. (2007). The epoch-greedy algorithm for contextual multi-armed bandits. In *Proceedings of the 20th international conference on neural information processing systems* (pp. 817–824).

Lau, H. C., & Zhang, L. (2003). Task allocation via multi-agent coalition formation: Taxonomy, algorithms and complexity. In *Proceedings. 15th ieee international conference on tools with artificial intelligence* (pp. 346–350).

Ma, K.-C., Ma, Z., Liu, L., & Sukhatme, G. S. (2018). Multi-robot informative and adaptive planning for persistent environmental monitoring. In *Distributed autonomous robotic systems* (pp. 285–298). Springer.

Narayan, S. (1997). The generalized sigmoid activation function: Competitive supervised learning. *Information sciences*, *99*(1-2), 69–82.

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, *3*(1), 88–97.

Ross, S., Gordon, G., & Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 627–635).

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming* (pp. 417–431).

Shiroma, P. M., & Campos, M. F. (2009). Comutar: A framework for multi-robot coordina-

tion and task allocation. In *2009 ieee/rsj international conference on intelligent robots and systems* (pp. 4817–4824).

Taylor, M. E., Suay, H. B., & Chernova, S. (2011). Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th international conference on autonomous agents and multiagent systems-volume 2* (pp. 617–624).

TREATY, N. A. (2008). Multi-robot systems in military domains. *NATO Research and Technology Organisation: Neuilly-sur-Seine, France*.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, *60*(3), 611–624.

Wicke, D., Freelan, D., & Luke, S. (2015). Bounty hunters and multiagent task allocation. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems* (pp. 387–394).

Zheng, J., Liu, S., & Ni, L. M. (2014). Robust bayesian inverse reinforcement learning with sparse behavior noise. In *Proceedings of the twenty-eighth aaai conference on artificial intelligence* (pp. 2198–2205).

Ziebart, B. D., Maas, A. L., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Aaai* (Vol. 8, pp. 1433–1438).